A dynamic-balanced scheduler for Genetic Algorithms for Grid Computing

A.J. SÁNCHEZ SANTIAGO[2], A.J. YUSTE[1], J.E. MUÑOZ EXPÓSITO[1], S. GARCÍA GALÁN[1], J.M. MAQUEIRA MARÍN[2], S. BRUQUE[2]
[1] Telecommunication Engineering Department
[2] Business Administration and Accounting Department
University of Jaén
Alfonso X El Sabio, 28. Linares, Jaén. SPAIN
{ajsantia, ajyuste, jemunoz, sgalan, maqueira, sbruque}@ujaen.es

Abstract. *The new paradigm of distributed computation, grid computing, has given rise to a large amount of research on resource scheduling. Unlike the distributed computation, grid computing uses heterogeneous resources, for what grid computing entails new challenges as the adaptation of parallel algorithms before developed for homogeneous resources cluster to the dynamic and heterogeneous resources. In this paper we present a dynamic-balanced scheduler for grid computing that solves two typical kinds of problems of grid computing, using for them the cycles of some resources of the grid. The first problem is based on iterative tasks that usually appear in optimization problems. The second problem is a directed acyclic graph (DAG) problem. Experimental results using dynamic-balanced scheduler show that it is possible to obtain an improved use of the resources in the grid. This strategy enables to adapt the length of a task to the computing capacity of each resource at any given moment. Furthermore, this scheduling strategy enables to execute all the tasks in a shorter time.*

Key-Words: - Grid computing, dynamic-balanced scheduler, genetic algorithm, optimization problem.

1 Introduction

Over the past few years, the development of high speed networks, increasing overall computing capacity and low cost of storage devices have given rise to a new paradigm of distributed computation: grid computing [1].
A Grid System, fig. 1, is made up by a set of Virtual Organizations (VO). A VO comprises a set of independent organizations that share heterogeneous resources to achieve a common mission, but that are not limited to an alliance among firms. Interaction among members of the virtual organization is mainly done through computer networks.

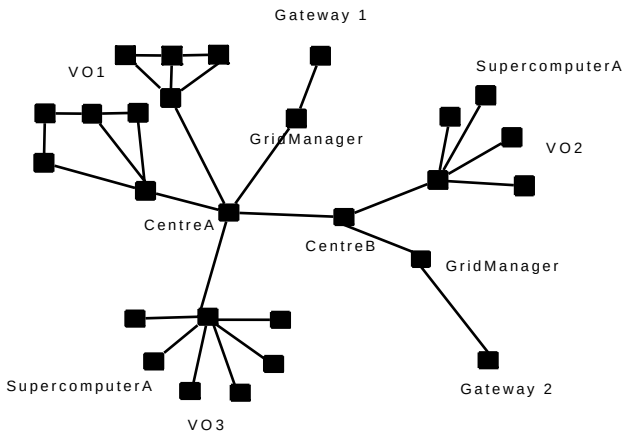

Figure 1. A Grid System

Unlike traditional distributed computation, which usually uses a geographical limited area, a grid system can be run throughout several organizations that utilize heterogeneous resources in a large geographical area.
Despite of its advantages, Grid computing also entails many new challenges, like the adaptation of parallel programs previously developed for homogeneous resources clusters to the dynamic and heterogeneous Grid resources with minimal intrusion into the code or resource scheduling. In a grid environment, the scheduler is the manager of the workflow, acting as intermediary between the user and the distributed resources, thus concealing the complexity of the grid system [2]. The scheduler identifies available resources, negotiates and organizes their use, decides when to execute the tasks and eventually organizes the results.
Overall, a scheduler makes a decision on when to execute a task in a given resource. This assignment is aimed to maximize (or minimize) an objective function, such as the time of execution, number of tasks not executed, execution costs, etc. Most of the recent literature treats the length of a task as a constant [3], which can not change during execution. This assumption for a grid, which is a highly dynamic environment, can make the scheduling algorithms simple and efficient; but can undermine the scheduling algorithms applicability in some

situations.
At the time of scheduling, the workload of the resources in the grid may be unknown. Furthermore, this workload may change over time, as the tasks start and finish periodically. Ignoring such uncertain factors may result in an increasing number of scheduling errors.
Nowadays many applications have been developed for traditional homogeneous parallel systems. Adapting these applications, developed for homogeneous environments, to dynamic heterogeneous resources entails a new challenge which has to do with keeping a high level of application efficiency. An adequate workload optimization method should take into account two main characteristics:

(1) Application characteristics, such as memory requirements, data being transferred between processes, application communication structures, hard disk or other I/O activity, etc.

(2) Resource characteristics, such as memory of the resources and computational capacity, network bandwidth, disk I/O speed, and the heterogeneity level of the resources randomly assigned to the application by the Grid resource broker.

The workload optimization method should be:

(a) Self-adapting and flexible with respect to the type of application.

(b) Computationally inexpensive; i.e. not to induce a large overhead on the application performance.

(c) Should not require significant modifications in the code.

In sum, the load balancing should be dynamic and fully automated since the complexity of the Grid environment should be hidden from end-users.
Traditionally there are two approaches that have proved useful for workload balancing in parallel applications: (1) to carefully calculate the distribution of the workload, taking into account all the properties of the environment and application (a time and resource consuming task requiring expert knowledge of the application structure and algorithms being used); and (2) distributing the workload in a straight forward way, optimizing the processing capacity of the resources.
Our aim, focused on computational grid, consists of building a scheduler for a SME (Small-Medium Enterprise) that allows to use the resources' idle cycles to execute tasks, which in a local machine would take several days, whereas in a grid system can only take a few hours.
We will develop a scheduler that can be used by the company's network managers, in such a way that is not necessary for them to obtain additional training or hire new employees.
With the scheduler we propose, we will minimize the overall response time for the entire workflow, while operation costs remain unchanged since the firm can use its own idle resources to run the grid.
Our scheduler splits the total workflow according to the idle cycles of each resource, so that all the resources can finish each task at the same time. The new scheduler is easy to use, is flexible and only needs an estimation of the process capacity of each resource as the main input. The simulations carried out in this research show that our algorithm improves others schedulers presented in prior literature.
This document is organized as follows: in section 2 we analyze recent work in grid resource scheduling. In section 3 we describe the dynamic-balanced scheduler for typical problems of grid computing. In Section 4 we show the results of the simulation using [4]. Finally, in section 5 we present the conclusions and future directions.

2 Background

Resource scheduling for tasks in a grid environment has been extensively studied in recent years. Some works focus on the scheduling of a set of independent tasks using scheduling techniques such as genetic algorithms [5], data mining [6] or swarm intelligence [7-8]. Other works have studied scheduling technologies using economic/market-based modelss [9].
More recent works consider data dependencies between tasks, which are threaded as a workflow with transitions indicating direct data dependencies. To minimize the complexity of scheduling a workflow, a workflow is partitioned into a set of subworkflows, each being assigned to a resource. In [10], Duan et al. proposed a workflow partitioning algorithm based on a set of rules that intend to minimize the data flow between subworkflows. Spooner et al. [11] proposed a two level algorithm that consisted of one global scheduler and a number of local schedulers, one for each grid resource.
Another research line leaves out the assumption that each grid resource must be a cluster. Instead, a grid resource is simply considered as the one that is

capable of providing services, computation, and/or storage. Some researches propose to determine the grid resources for all tasks in a workflow before the workflow is started, which is referred to as static resource scheduling in subsequent discussion. In [12], a pair of a task (t) and a grid resource (r) receives a rank which is a linear combination of computing and data transfer time for executing t on r. Three algorithms, namely min-min, max-min, and sufferage, were proposed to minimize the makespan of the entire workflow based on the ranks.

Considering the fact that grid may be highly dynamic, some researches propose to perform resource scheduling only for tasks that are ready to execute, referred to as dynamic resource scheduling. Blythe et al. [13] adopted min-min strategy for both static and dynamic resource scheduling and concluded that the static resource scheduling is more suitable for data-intensive workflows, while dynamic resource scheduling performs better for computation-intensive workflows.

Weiss and Pinedo [14] have shown that the SEPT (Short Expected Processing Time) strategy, which gives priority to jobs with shortest expected processing times, and the LEPT (Long Expected Processing Time) strategy, which gives priority to jobs with longest expected processing times, can be used to minimize expected flow time and expected makespan respectively, when the processing time of each job follows an exponential distribution. However, in a real grid environment, the processing time of each task by a resource may not follow an exponential distribution.

In these works the length of a task from a workflow is known in advance and the optimizations are based on an in advance predetermined workflow. Conversely, we experiment in this work with several workflows in which the size of each task is different.

3 The proposed algorithm

The design space for Grid Schedulers in general is very rich. First, it depends on what objective function the user wants to minimize or maximize – examples being minimizing overall job completion time, minimizing communication time and maximizing resource utilization or throughput. Second, it depends on how the job requirements, job performance models, and Grid resource models are specified and used. The scheduler must also carefully choose between different implementations of user authentication, allocation, and reservation. Other choices include scheduling application components for single or multiple users and whether rescheduling or re-planning is required.

There are usually two types of problems in task scheduling: an optimization problem and a directed acyclic graph (DAG) problem, fig. 2. In this figure we show an example of a simple DAG with 10 tasks. Tasks 5, 6 and 7, cannot start until the previous tasks 2, 3, 4 has finished. For the last task, 10, it cannot begin until tasks 8 and 9 are finished, even if task 7 was finished before.

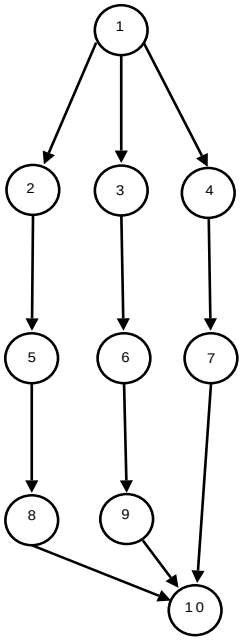

Figure 2. A DAG

With the adaptive dynamic-balanced scheduler we try to validate two problems: on the one hand a genetic algorithm for solving optimization problems which has a set of independent workflows, and on the other hand a DAG problem. In both cases we have used CPU idle cycles of some resources, which a company has in several networks, minimizing the time of execution of each workflow.

Each task requires some data as input and output and may be executed in one of a several resources, which satisfy the overall objective.

3.1 The model

We develop a general model to verify the validity of our scheduler. Our model is made up by a set of tasks, grouped in a workflow. A workflow will not begin next iteration unless the prior one had been finished. A typical example of this is produced in processes of optimization, like genetic algorithms, in which until a generation has not been fully evaluated, it is not possible to evaluate the following generation.

In order to formulate an integrated model, the following parameters are introduced:

Parameters:

n: numbers of resources
iter: numbers of iterations
N: load for each iteration
N_i: task for each resource
μ_i: instructions per second (MIPS)
Ti: time of iteration
OT: overall time of a workflow

In our scheduling, we have chosen our objective function: minimize overall job completion time. It is only necessary to count the maximum time of execution from the worst relation task–resource. This time can be worked out with the following mathematical equation (1).

$$OT = \sum_{i=1}^{iter} MAX_i \left(\frac{N_i}{\mu_i} + Ts_i + Tr_i \right) \quad (1)$$

The time of an iteration (2) is given by the longest time needed to execute a task in the worst available resource.

$$T_i = \sum_{k=1}^{tasks} MAX_i \left(\frac{N_i}{\mu_i} + Ts_i + Tr_i \right) \quad (2)$$

Where Ts_i is the time that takes sending a task to the resource that is going to execute it and Tr_i the time that takes receiving the result from a task. We consider the time of sending and receiving a task negligible, because if this time is higher than the time of execution of the task we do not send it to the grid. Thus, the time of one particular iteration can be simplified by (3):

$$T_i \cong MAX_i \frac{N_i}{\mu_i} \quad (3)$$

To minimize this T_i we assign a proportional load to each following resource, this way we do not depend on the resource that offers a worse performance:

$$N_i = N \times \frac{\mu_i}{\sum_{i=1}^{n} \mu_i} \quad (4)$$

Thus, the innovation in this study lies in that we carry out a previous study of the idle cycles of all the resources we have, adapting the length of a task is according to the total idle cycles.

With this proposal we obtain two results: on the one hand we carry out a better load balancing for the resources, assigning higher loads to the resources presenting a higher availability and, on the other hand, we reduce the time of execution of the workflows.

3.2 An execution

In the following figures we show the process of execution of iteration in our model by a simple example.
In the first step, fig. 3, we can identify three resources of our Grid System. The scheduler asks the resources, by the function Init_Job(I_J), for the μi idle that they have. Every resource sends to the scheduler the number of idle cycles that it has available for this iteration.

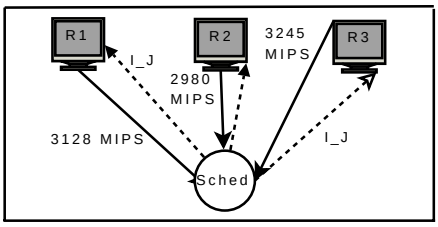

Figure 3. Step 1.Get MIPS idle from each resource

In the second step, a workflow enters the Grid System, and the scheduler realizes his work, splits this workflow according to the idle cycles of each resource, fig 4.

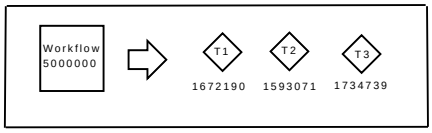

Figure 4. Step2 .Split workflow according idle MIPS of each resource

In the next step, fig. 5, each resource executes his task, finishing the execution in a similar time for all the resources, so we do not depend on the execution of the worst resource.

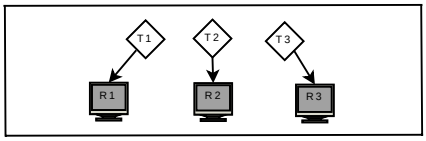

Figure 5. Step3. Execute the tasks in the corresponding resources

In the last step, the scheduler obtains the results for this iteration and the idle MIPS that they have for the execution of the next iteration.

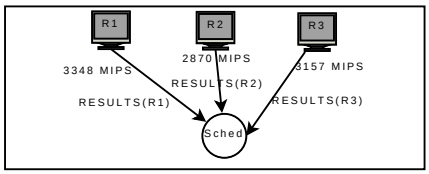

Figure 6. Step 4. Get results and the idle MIPS for the next iteration

As seen during the example, our scheduler is highly dynamic, because after every execution, the scheduler is able to re-obtain the actual number of idle cycles.

4 Experimental results

Due to the difficulties associated to real tests, the benefits of this scheduler have been verified through the use of simulations. Simulations offer big benefits opposite to other methods that are currently used to measure a system's behavior:
a. Simulations allow experimentation in the system without interrupting the activity of the system and therefore avoiding potential damages.
b. Simulations are independent on the process or real system. Therefore, it is possible to use simulations in different stages before their actual implementation in real settings.
c. Simulations improve the comprehension of the behaviour of the system since they provide numerical results.
d. Simulations allow to carry out comparative analyses of different scenarios.
Therefore, it was necessary to develop our scheduler using the GridSim toolkit [4]. GridSim is a simulation toolkit for resource modelling and application scheduling in parallel and distributed computing systems.
Table 1 shows the type of the resources we have used in our analysis. The MIPS for each PE (Processing Element) is the maximum MIPS for each resource. This value can change according to the use of CPU's cycles that the user of the resource needs at any given moment, modelled by a uniform distribution. We assume that delay of transmission is negligible compared with the time of execution of a task.

Resourc.	R1	R2	R3	R4	R5
S.O.	Win	Unix	Win	Win	Win
Arq.	IBM	Solaris	IBM	IBM	IBM
P.E.	1	1	1	1	1
MAX MIPS	2000	2500	3000	3500	4000

Table 1. Type of resources

We compare our proposal with other, more traditional schedulers, space-shared and time-shared policies. FCFS (First-Come First-Served) [15] is a time-shared algorithm. This is one of the simplest, executes the workflows to completion in the order they are submitted. It has a big drawback; if a resource has few idle cycles to offer, the execution of the remaining workflows will turn out slow, since it is necessary to fulfil all tasks of a workflow to execute the following workflow.
RR (Round Robin) [16] is a time-shared algorithm; it is one of the oldest, simplest, fairest and most widely used scheduling algorithms, designed especially for time-sharing systems. A small unit of time, called time slices or quantum is defined. All runable processes are kept in a circular queue. The scheduler goes around this queue, allocating the CPU to each process for a time interval of one quantum. New processes are added to the tail of the queue. The scheduler picks the first process from the queue, sets a timer to interrupt after one quantum, and dispatches the process. If the process is still running at the end of the quantum, the CPU is pre-empted and the process is added to the tail of the queue. If the process finishes before the end of the quantum, the process itself releases the CPU voluntarily. In either case, the CPU scheduler assigns the CPU to the next process in the ready queue. Every time a process is granted the CPU, a context switch occurs, which adds overhead to the process execution time.
The experiments can be divided in two groups. Firstly, we have executed a workflow based on independent tasks that usually appear in optimization problems. Secondly, we executed a DAG problem.

4.1 An optimization problem

The Genetic Algorithm (GA) is a method for solving optimization problems that is based on natural selection, the process that drives biological evolution. The GA repeatedly modifies a population of individual solutions. At each step, the GA randomly selects individuals from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population evolves toward an optimal solution. The GA can be applied to solve optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, non-differentiable, stochastic, or highly nonlinear as our problem. GA is frequently used in research like in network communications [17-18], for speech/music discrimination [19] or for partitioning problems in code sign [20].
We have carried out several simulations by changing the number of iterations; each iteration being made

up by a workflow and the available resources. We defined the length of a workflow large enough, to take advantage of a system grid, and moderately small in order to not overload the less efficient resources.
We have developed four experiments with 25, 50, 75 and 100 resources that we explain in detail below.

4.1.1 First experiment

The first experiment was made up by 25 resources and 100, 500 and 1000 iterations. The times are shown in table 2, 3 and 4. We can see in fig. 7 that the execution time was getting shorter for the RR and balanced algorithms.

Nº iterations=100 Workflow length= 500000000 Nº resources= 25	
Algorithm	Time (sec.)
FCFS	815681,27
RR	1028379,16
Balanced	751859,56

Table 2. 100 iterations and 25 resources

Nº iterations =500 Workflow length= 500000000 Nº resources= 25	
Algorithm	Time (sec.)
FCFS	4070361
RR	5108821,63
Balanced	3761625,95

Table 3. 500 iterations and 25 resources

Nº iterations=1000 Workflow length= 500000000 Nº resources= 25	
Algorithm	Time (sec.)
FCFS	8153256,08
RR	10218636,6
Balanced	7532137,76

Table 4. 1000 iterations and 25 resources

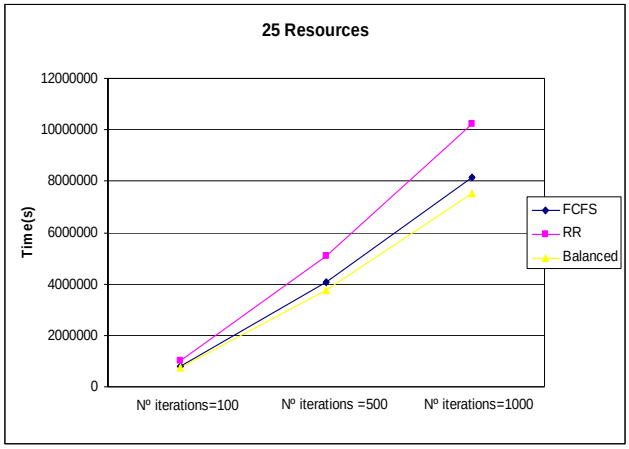

Figure 7. Experiment with 25 resources and Workflow length= 500000000

4.1.2 Second experiment

Table 5, 6 and 7, and fig. 8 show a comparison between the time for the three schedulers for 50 resources and 100, 500 and 1000 iterations. Fig. 8 depicts that the higher the number of iterations, the larger the difference of execution time, which implies that FCFS and RR degenerate very quickly as the number of iterations grows.

Nº iterations =100 Workflow length= 500000000 Nº resources= 50	
Algorithm	Time (sec.)
FCFS	526393,32
RR	548574,52
Balanced	402459,76

Table 5. 100 iterations and 50 resources

Nº iterations =500 Workflow length= 500000000 Nº resources= 50	
Algorithm	Time (sec.)
FCFS	2637932,84
RR	2740363,3
Balanced	2016518,16

Table 6. 500 iterations and 50 resources

Nº iterations =1000 Workflow length= 500000000 Nº resources= 50	
Algorithm	Time (sec.)
FCFS	5269542,88
RR	5487507,1
Balanced	4031486,28

Table 7. 1000 iterations and 50 resources

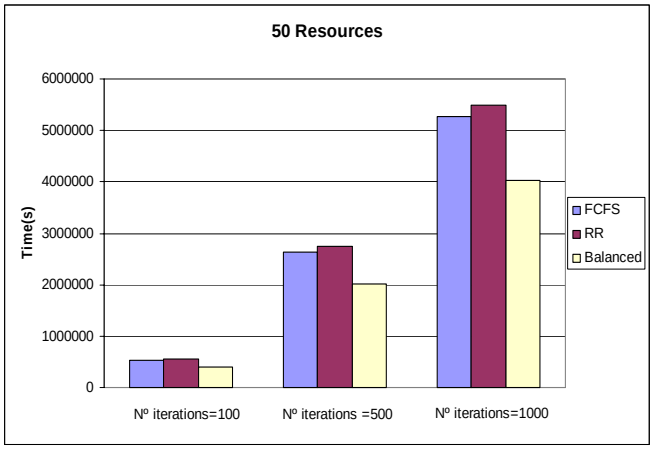

Figure 8. Experiment with 50 resources and Workflow length= 500000000

4.1.3 Third experiment

Tables 8, 9 and 10, and fig. 9 exhibits time of execution for 75 resources and 100, 500 and 1000 iterations. Like in the aforementioned experiments, our balanced scheduler offers better results.

Nº iterations =100 Workflow length= 500000000 Nº resources= 75	
Algorithm	Time (sec.)
FCFS	352608,2
RR	436609,265
Balanced	313611,72

Table 8. 100 iterations and 75 resources

Nº iterations =500 Workflow length= 500000000 Nº resources= 75	
Algorithm	Time (sec.)
FCFS	1760069,48
RR	2173044,77
Balanced	1567549,15

Table 9. 500 iterations and 75 resources

Nº iterations =1000 Workflow length= 500000000 Nº resources= 75	
Algorithm	Time (sec.)
FCFS	3522367,51
RR	4357151,66
Balanced	3136368,64

Table 10. 1000 iterations and 75 resources

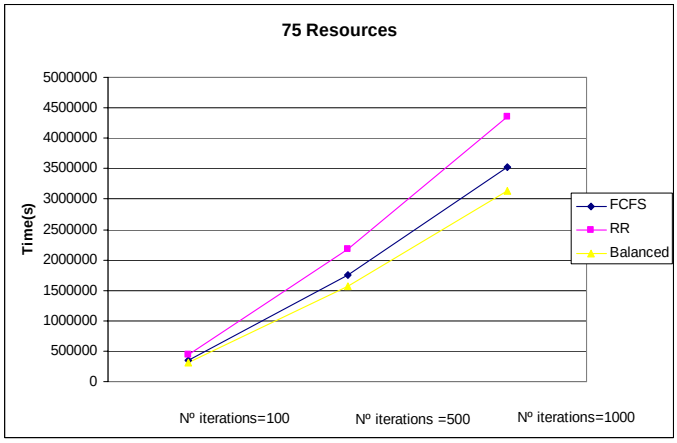

Figure 9. Experiment with 75 Resources and Workflow length= 500000000

4.1.4 Fourth experiment

The last experiment that we carried out, tables 11, 12, 13 and fig. 10, with 100 resources, offer the minimum execution time of all our tests.

Nº iterations=100 Workflow length= 500000000 Nº resources= 100	
Algortihm	Time (sec.)
FCFS	350501,83
RR	343266,82
Balanced	270239,52

Table 11. 100 iterations and 100 resources

Nº iterations =500 Workflow length= 500000000 Nº resources= 100	
Algortihm	Time (sec.)
FCFS	1748339,75
RR	1722092,89
Balanced	1350322,72

Table 12. 500 iterations and 100 resources

Nº iterations=1000 Workflow lengh= 500000000 Nº resources= 100	
Algortihm	Time (sec.)
FCFS	3502678,19
RR	3438388,70
Balanced	2698851,20

Table 13. 1000 iterations and 100 resources

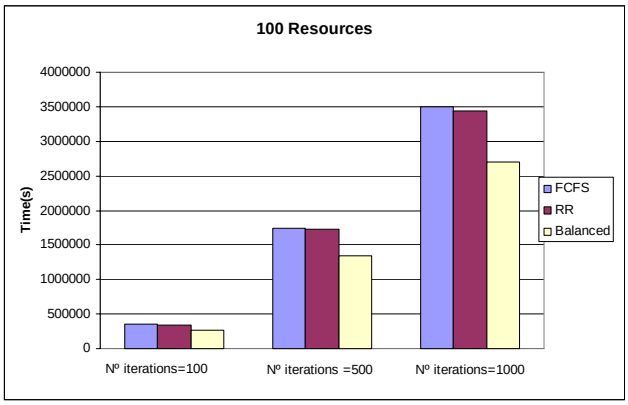

Figure 10. Experiment with 100 Resources and Workflow length= 500000000

4.1.5 Summary of results

The previous examples that we have carried out in this study show a comparison of times from the point of view of the iterations, meanwhile this last experiment compares execution times according to the resources being used.
In fig. 11 we show a comparison for 100 iterations and 25, 50, 75 and 100 resources. The dynamic-balanced scheduler we proposed offers again better results than any other two schedulers for every mix of resources.

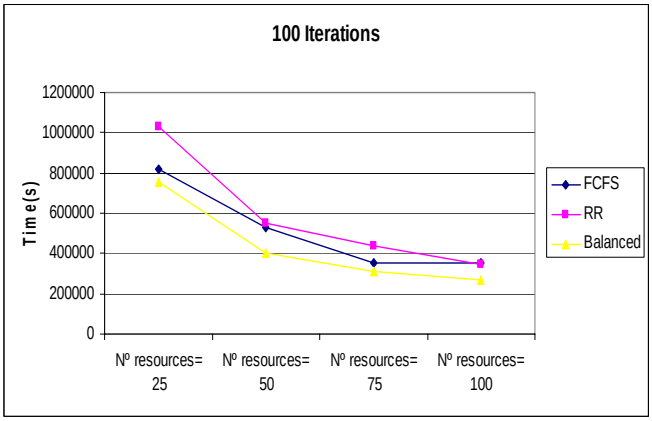

Figure 11. Experiment with 100 iterations and Workflow length= 500000000

Fig. 12 exhibits the execution time for 500 iterations and 25, 50, 75 and 100 resources. At any given combination of resources our balanced scheduler obtains the shorter execution time.

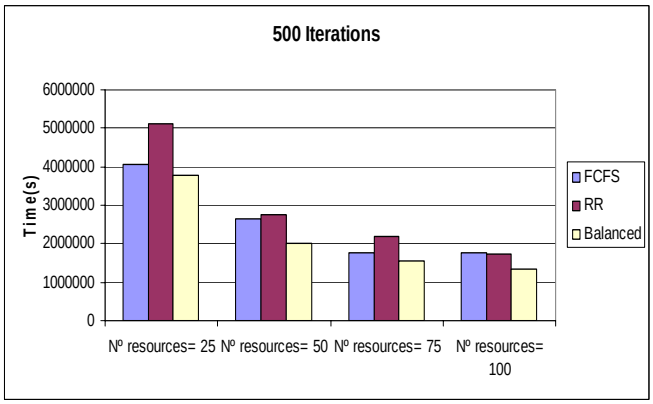

Figure 12. Experiment with 500 iterations and Workflow length= 500000000

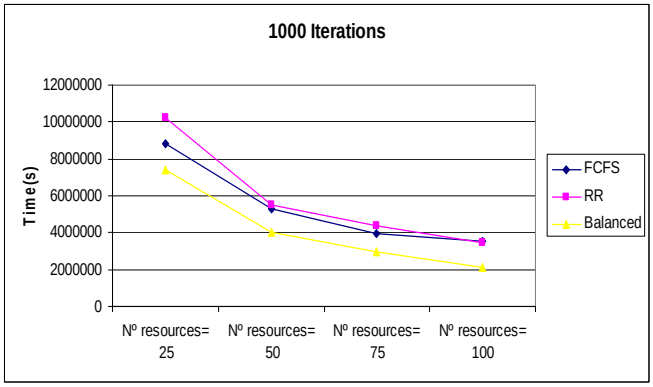

Figure 13. Experiment with 1000 iterations and Workflow length= 500000000

Fig. 13 exhibits the execution time for 1000 iterations and 25, 50, 75 and 100 resources.

4.2 A DAG problem

In this experiment we executed a standard DAG, which consisted of a protein annotation workflow (Fig. 14), [9].

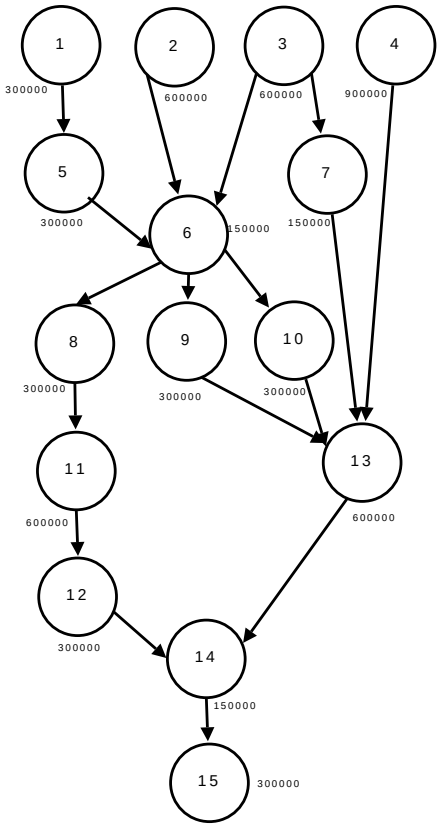

Figure 14. Protein annotation workflow

This experiment, protein annotation workflow, was executed with the parameters shown in Table 14.

Resources	100
S.O.	Win
Arq.	IBM
MAX MIPS	6000
BaudRate(Mb)	100
Prop. Delay(ms)	10

Table 14. Parameters for Grid System for protein annotation workflow

During this second experiment, our dynamic-balanced scheduler improves the time of execution of the FCFS and RR schedulers. Results are presented in Table 15.

Prot. annot. workflow	Time (sec)
Balanced	2702.52
FCFS	3741.82
RR	3916.23

Table 15. Execution time; protein annotation workflow

5 Conclusions and future work

A grid environment is usually heterogeneous in nature in the real world at least for the different computing speeds at different participating sites. The heterogeneity presents a challenge for effectively arranging load sharing activities in a computational grid. This paper explores job scheduling and allocation issues in heterogeneous computational grids when a task, during the scheduling process, cannot fit in any single site in the grid.

We have proposed a balanced scheduler in a grid environment. With this technique we have fulfilled three objectives: Firstly execution time has been shortened, secondly we have obtained a better loading distribution among resources in the grid and thirdly we have used idle, non cost-increasing resources.

As future research directions, we will conduct additional experiments to comprehensively compare other scheduling strategies using fuzzy logic that would enable us to set up a utility grid structure. These utility models are very effective in shared inter-organizational settings to increase computing capacity whilst keeping security and reducing operating costs.

6 Acknowledgments

This work has been financially supported by the Andalusian Government (Research Project P06-SEJ-01694).

References

[1] I. Foster and C. Kesselman (editors), *The Grid: Blueprint for a new Computing infrastructure*. Morgan Kaufmann Publishers, San Francisco, USA, (1999)

[2] R. Buyya, D. Abramson, and J. Giddy, Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, *4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region* (HPC ASIA 2000), Vol 1,2000, pp 283.

[3] Reakook Hwang, Mitsuo Gen, Hiroshi Katayama. A comparison of multiprocessor task scheduling algorithms with communication costs. *Computers and Operations Research* Volume 35, *Issue 3*, 2008, pp 976-993.

[4] Anthony Sulistio, Uros Cibej, Srikumar Venugopal, Borut Robic and Rajkumar Buyya A Toolkit for Modelling and Simulating Data Grids: An Extension to GridSim. *Concurrency and Computation: Practice and Experience (CCPE)*, Wiley Press, New York, USA 2007.

[5] Cao, J., Spooner, D.P., Jarvis, S.A., Nudd, G.R.: Grid load balancing using intelligent agents. *Future Generation Comput. Syst.* Vol. 21(1), 2005, pp 135–149.

[6] Meiqun Liu, Kun Gao, Zhong Wan, A Novel Architecture for Data Mining Grid Scheduler, *WSEAS Transactions on Systems* Issue 1, Volume 7, 2008 pp 373-383.

[7] Wang Da-Zhen, Zhan Jun-Shan, Wan Fang, Zhu Lei., A Dynamic Task Scheduling Algorithm in Grid Environment. *WSEAS Transactions on Computers*, Issue 7, Volume 5, 2006, pp 1632-1634.

[8] Reche López, P., Gómez González, M., Ruiz Reyes, N., Jurado F., "Optimization of biomass fuelled systems for distributed power generation using Particle Swarm Optimization". *Electric Power Systems Research.* Issue 8, Vol. 78, 2008, pp. 1448-1455.

[9] Yu, J. and Buyya, R., A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms, Workshop on Workflows in Support of Large-Scale Science, *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC 2006, IEEE CS Press, Los Alamitos, CA, USA)*, 2006, Paris, France.

[10] Duan, R., Prodan, R., Fahringer, T.: Run-time optimisation of grid workflow applications, *7th IEE/ACM International Conference on Grid Computing*, 2006, pp. 33–40.

[11] Spooner, D.P., Cao, J., Jarvis, S.A., He, L., Nudd, G.R.: Performance-Aware Workflow Management for Grid Computing. *The Computer Journal*, Vol 48, 2005, pp 347–357.

[12] Mandal, A., Kennedy, K., Koelbel, C., Marin, G., Mellor-Crummey, J., Liu, B., Johnsson, L.: Scheduling strategies for mapping application workflows onto the grid. *Proceedings of the 14th International Symposium on High Performance Distributed Computing (HPDC 2005)*, 2005, pp. 125–134.

[13] Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A., Kennedy, K.: Task scheduling strategies for workflow-based applications in grids. *Proceedings of the Cluster Computing and Grid 2005 (CCGrid 2005)*, 2005, pp. 759–767.

[14] Weiss, G., Pinedo, M.: Scheduling Tasks with Exponential Service Times on Non- Identical Processors to Minimize Various Cost Functions. *Journal of Applied Probability*, 1980, pp 187–202.

[15] Domenico Ferrari. Real-time communication in an internetwork. *Journal of High Speed Networks*, Vol. 1, 1992.

[16] Tanebaum, A.S., *Modern operating systems*. Prentice-Hall, 1993.

[17] Antonio J. Yuste, Francisco David Trujillo, Alicia Triviño, Eduardo Casilari, An adaptive gateway discovery for mobile ad hoc networks. *Proceedings of the 5th ACM international workshop on Mobility management and wireless access, MobiWac '07,* 2007, pp 159-162.

[18] J. A. Fernández Prieto, Juan R. Velasco Pérez. Adaptive Genetic Algorithm control parameter optimization to verify the network protocol performance. *12th International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems (IPMU)*, Torremolinos(Spain) pp. 785-791, 2008.

[19] J.E. Muñoz, S. García Galán, N. Ruiz-Reyes, P. Vera-Candeas, Audio coding improvement using evolutionary speech/music discrimination. *IEEE International Conference on Fuzzy Systems (FUZZY-IEEE 2007)*, London, pp 1-6, 2007.

[20] Abderrazak Henni, Mouloud Koudil, Karima Benatchba, Hassane Oumsalem, Kamel Chaouche, A parallel environment using taboo search and genetic algorithms for solving partitioning problems in codesign, *WSEAS Transactions on Systems*, Issue 1, Volume 3, 2004, pp 8-13.